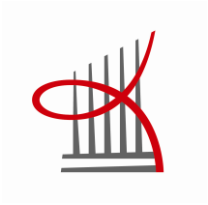


XNA GAME STUDIO 4.0

Mika Parkkola

Opinnäytetyö
Joulukuu 2012
Tietotekniikka
Ohjelmistotekniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Ohjelmistotekniikka

MIKA PARKKOLA:
Xna game studio 4.0

Opinnäytetyö 65 sivua, joista liitteitä 37 sivua
Joulukuu 2012

Tämän työn tarkoituksena on opiskella ja tutkia XNA Game Studion toimivuutta. Työssä tutustutaan eri osa-alueisiin XNA:ssa ja kuinka sillä saadaan tehtyä interaktiivinen pelisovellus.

XNA on Microsoftin kehittämä Cross-platform, jonka esiversio nimeltään Windows Game SDK julkaistiin jo Windows95 käyttöjärjestelmälle. Nykyinen 4.0 versio on julkaistu vuonna 2010.

XNA antaa pelinkehittäjälle valmiin alustan, jonka avulla pystytään kehittämään peliä suoraan. Sen valmiit luokat ja metodit antavat tehokkaat työkalut pelin kehittämiseen. Tällöin pelisovelluksen suunnittelussa ei tarvitse ottaa huomioon alustan vaatimia rutiineja ohjelman suorittamiseksi.

XNA:n ohjelmointikieli on Mircosoftin kehittämä tehokas C# kieli. Ohjelman kehityksessä täytyy ottaa huomioon tekeekö ohjelman Windowsille, XBOX:lle vai Windows Phonelle. Riippuen kohdealustasta joudutaan vain miettimään pelaajan antamien syötteiden lukeminen ja niihin reagointi. Muut osa-alueet pelisovelluksesta ovat samoja kohdealustasta riippumatta.

Asiasanat: XNA, pelikehitys, microsoft, XBOX360, WP

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Computer science
Software Engineering

MIKA PARKKOLA:
Xna game studio 4.0

Bachelor's thesis 65 pages, appendices 37 pages
December 2012

The purpose of this thesis was to learn XNA Game Studio and it's principles. This thesis deals with the different aspects of XNA for and how it can be up to an interactive game application.

XNA is developed by Microsoft for game developers to help them creating games for Windows, XBOX360 and Windows Phone 7 and later versions. Its classes and methods provide powerful tools for game development so developers does not need to take into account the substrate required by routines for execution of the program.

XNA is developed by C# coding language developed by Microsoft. In designing the application the developer has to take account the target platform, Windows, XBOX360 or WP7, because inputs from the player are read differently. Other aspects of the application are irrelevant from the target platform..

Key words: XNA, game development, microsoft, XBOX360, WP

SISÄLLYS

TERMIT

XNA	Microsoftin kehittämä pelinkehitys työkalu.
C#	Microsoftin kehittämä ohjelmointikieli.
Framework	Ohjelmistokehys.
Model	Olio, joka sisältää tiedot kolmiulotteisesta kuviosta.
Gamepad	Konsolien ohjaamiseen tarkoitettu ohjain.
Shader	Ohjelmakokonaisuus, joka suoritetaan näytönohjaimen prosessorissa.
HLSL	Shadereiden ohjelmointia varten kehitetty ohjelmointikieli.
Windows	Microsoftin kehittämä käyttöjärjestelmä.
Windows 95 /Vista / XP	Microsoft Windowsin eri versioita.
XBOX	Microsoftin valmistama pelikonsoli.
Windows Phone	Microsoftin kehittämä käyttöjärjestelmä mobiililaitteisiin.
DirectX	Microsoftin kehittämä ohjelmakirjasto graafisten sovellusten tekemiseen Windowsille.
.NET	Microsoftin kehittämä ohjelmistokomponenttikirjasto.
Microsoft LIVE	Internetin yli toimiva sosiaalinen pelipalvelu.
Visual Studio	Microsoftin kehittämä sovellusten kehitystyökalu-kokonaisuus.
ZUNE	Microsoftin digitaalisen musiikin brändi, joka ensin oli tuoteperhe musiikkisoittimille, joka lopetettiin lokakuussa 2011. Nykyisin ZUNE on musiikin latauspalvelu.
Renderöinti	Matemaattinen toiminto, joka laskee näytölle piirrettävän kuvan tiedot.
Cross-Platform	Järjestelmäriippumaton ohjelma, jota pystytään suorittamaan monella eri käyttöjärjestelmällä muuttamatta sitä erikseen sopivaksi.
Malli	XNA:ssa kätetty kolmiuloitteisen kohteen mallinnustiedot sisältävä luokka.
Quaternion	Struktuuri, jossa on tallennettuna neljä uloitteinen vektori, jonka avulla pystytään pyörittämään mallia tehokkaasti 3D-maailmassa.

1 JOHDANTO

XNA Game Studio on Microsoftin DirectX:n ja .NET frameworkin päälle kehittämä pelinkehitys ohjelmistokomponenttikirjasto. Se on suunniteltu vapauttamaan kehittäjät tekemään suoraan pelilogiikkaa, koska XNA suorittaa ohjelman perustan itsenäisesti. Kehittäjän ei tarvitse enää huolehtia ohjelman perusrungosta, syötteistä sekä muista peliohjelmointiin tarvittavista ajureista tai rutiineista.

XNA:lla pystytään tekemään joko pelkästään 2D tai 3D grafiikkaa, sekä näiden yhdistelmiä. Siihen on rakennettu kaikki tarvittavat toiminnot, kuten syötteen lukeminen peliohjaimilta, sekä äänen tuottaminen pelaajalle. Kehitys tapahtuu Microsoftin kehittämällä C# olio-ohjelmointikielellä.

Opinnäytetyön ensimmäisessä osiossa tutustutaan tarkemmin XNA-ohjelmoinnin historiaan sekä työkaluihin, joita kehittäjä tarvitsee pelin valmistuksen yhteydessä. Osiossa tutustutaan myös muihin pelin valmistuksessa käytettyihin työkaluihin.

Toisessa osiossa tarkastellaan pelin suunnittelun vaiheita, kuten sitä kuinka ideasta jalostetaan toteutettava peli ja miten se paloitellaan pienempiin osiin, jotka on helppo toteuttaa.

Kolmannessa osiossa keskitytään pelin tekniseen toteutukseen. Siinä peli käydään läpi osa osalta, näyttäen koodi esimerkein kuinka se on toteutettu.

Viimeisessä osiossa keskitytään pelin ääniin sekä sen visuaaliseen puoleen, kuten pelin elävöittämiseen äänillä sekä sopivilla tekstuureilla.

2 XNA GAME STUDIO

XNA Game Studio on Microsoftin alun perin vuonna 2006 julkaisema ohjelmistokomponenttikirjasto pelinkehittäjille. Sen tarkoituksena on vapauttaa pelintekijät kirjoittamasta toistuvia koodinosia, joita tarvitaan toimivan ohjelman toteuttamiseksi. Tällöin pelintekijät pystyvät keskittymään olennaiseen, eli suunnitteluun ja pelimekaniikan tekemiseen.

XNA:lla tehdyt pelit on käännettävissä Windows XP, Windows Vista, Windows 7, Windows Phone 7 käyttöjärjestelmille, sekä XBOX360 alustalle.

2.1 XNA historia

XNA:n historia alkaa jo vuodesta 1995, jolloin Microsoft julkaisi Windows Game SDK:n. Tämä nimettiin myöhemmin uudelleen, nykyään tutummaksi, DirectX ohjelmistoksi. DirectX kehittyi vuosien saatossa uudempi versio. Version 7 aikaan se sai toiminnallisuuksia, jotka sallivat muidenkin kuin C-kielen käyttämisen sen käsittelyssä. Versiossa 9 siihen sisällytettiin komponentti CLR:lle (Common Language Runtime), joka mahdollisti C# käytön DirectX:n käyttämiseen. (Wikipedia 2012)

Vuonna 2006 DirectX:n päälle kehitettiin XNA:n ensiversio, nimeltä XNA Game Studio Express. Versio on ollut alusta alkaen helppo käyttää ja tehokas ympäristö, joka mahdollisti pelien kehittämisen ja pyörittämisen Xbox 360 järjestelmässä. (Wikipedia 2012)

Tämän jälkeen XNA on kehittynyt tasaisesti eteenpäin tuoden uusia työkaluja pelinkehittäjille. Versiossa 2.0, joka julkaistiin vuonna 2007, asennettiin tuki Microsoftin LIVE-ympäristöön, joka mahdollistaa verkkopelaamisen. Tällöin myös mahdollistettiin se, että kehittäjien on mahdollista käyttää mitä tahansa Visual Studio 2005 versiota, eikä ainoastaan C# express versiota. (Wikipedia 2012)

3.0 versiossa XNA:han lisättiin tuki myydä sillä kehitettyjä pelejä LIVE-palvelussa, sekä siihen asennettiin ZUNE tuki. 3.1 versio lisäsi tuet Zune HD:lle, videolle, Xbox

Live- ryhmille sekä Avatareille, joita käytetään Xbox Live palvelussa kuvastamaan pelaajia. Näitä samoja Avatareja voidaan käyttää myös kuvastamaan pelajaa pelin sisällä. (Wikipedia 2012)

Nykyisessä, vuonna 2010 julkaistussa, 4.0 versiossa saatiin tuki uudelle Windows Phone 7 -käyttöjärjestelmälle. Kaikissa versioissa on kehitetty aikaisemmin jo XNA:han lisättyjen toimintojen toimintaa yksinkertaistaen niiden käyttöä. Mikrofonin käyttö, sekä dynaaminen audio, eli äänen luku pelin aikana äänilähteestä, ovat saatu nykyiseen versioon toimivaksi kokonaisuudeksi.

Uusin versio on jaettu kahteen osioon, Reach- sekä HiDef- osioon. Reach-versiossa käytetään ominaisuuksia, jotka soveltuvat kaikille alustoille. HiDef taas sisältää lisäominaisuuksia, joita ei pystytä käyttämään Windows Phone 7 alustalla, vaan Xbox 360 ja Windows (riippuen grafiikkakortin laadusta) aluistoilla. (Miller & Johnson 2011, 3)

2.2 Työkalut

XNA:n kehitys tapahtuu C#-kielellä. Se on Microsoftin vuonna 2001 kehittämä olio-ohjelmointikieli. Tämän työn tekemiseen on käytetty Microsoftin ilmaista, ei-kaupalliseen käyttöön tehtyä, Visual C# 2010 Express -työkalua. Se käyttää C#:in uusinta, 4.0 versiota, joka julkaistiin huhtikuun 12. päivä vuonna 2010.

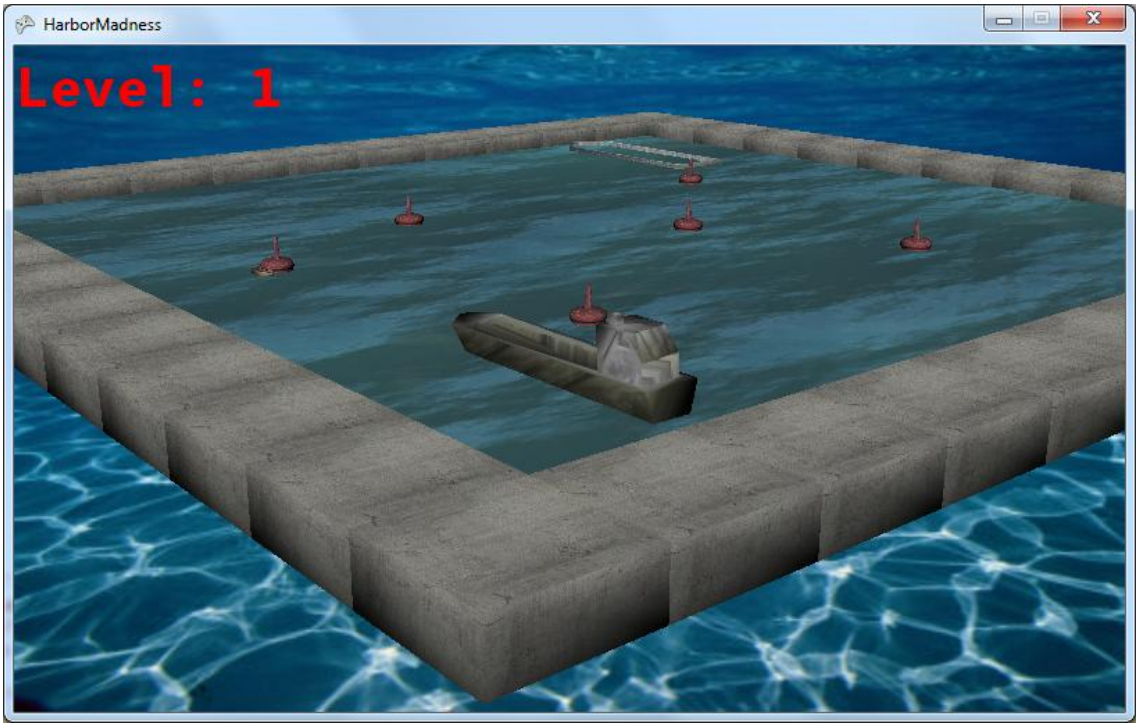
3D-mallien tekemiseen käytettiin MAXON:n kehittämää Cinema4D 3D-mallinnusohjelmistoa, joka on erittäin tehokas sovellus. Sillä tehdyt mallit saadaan tallennettua suoraan XNA:n ymmärtämään muotoon, jolloin mallien siirto peliin on erittäin yksinkertaista.

3 SUUNNITTELU

Hyvällä suunnittelulla säästetään aikaa ja sen vaatimia kehityskustannuksia. Hyvin kehitetty peli on helppo toteuttaa teknisesti. Tässä osiossa tutustutaan ja luodaan katsauksia tämän projektin eri suunnitteluvaiheisiin.

3.1 Peli-idea

Projektissa toteutettiin yläperspektiivistä kuvattu kevyt laivanohjauspeli, nimeltään Harbor Madness. Pelin ideana on ohjata tankkeria satamassa, saattaen se oikeaan laituriin. Jokainen pelin satama on ahtaampi ja vilkkaammin liikennöity kuin edellinen. Kuvassa 1 on esitelty pelin graafista käyttöliittymää ja pelinäkymää



KUVA 1. Kuva pelitilanteesta.

Pelaaja ohjaa laivaa säätämällä moottorien voimakkuutta ja suuntaa, sekä peräsimen asentoa. Pelaajan tulee ennakoinnin ja ohjaustaitojen avulla saattaa laiva turvallisesti laituriin, väistäen samalla mahdollisia toisia satamassa olevia laivoja tai muita esineitä.

Jokaisessa satamassa on tietty laituri, jonne laiva täytyy saada ohjattua. Kentän pääsee läpi, kun pelaaja on saanut laivan pysäytettyä laiturin viereen. Tämän jälkeen pelaaja voi siirtyä seuraavaan kenttään. Jos pelaaja ei onnistu tavoitteessaan, joudutaan kenttä aloittamaan uudelleen alusta.

3.2 Ominaisuudet

Peliin haluttiin oikeata fysiikkalaskentaa mallintamaan laivan käyttäytymistä pelissä, sekä huomioimaan veden mahdollinen vaikutus suhteessa laivan liikkumiseen. Peliin päätettiin toteuttaa seuraavat vaikuttavat elementit:

1. Jatkuvuuden laki: Laiva pyrkii jatkamaan kulkuaan, jollei siihen vaikuta liikerataa muuttavia voimia.
2. Moottori: Moottoreita ohjaa pelaaja, joilla tämä pyrkii vaikuttamaan laivan kulkusuuntaan toivotulla tavalla.
3. Veden vastus: Vesi vastustaa laivan kulkua hidastaen sitä tasaisella voimalla. Näin laiva ei siis voi jatkaa kulkuaan loputtomiin, vaan pysähtyy tietyn ajan kuluessa itsekseen.

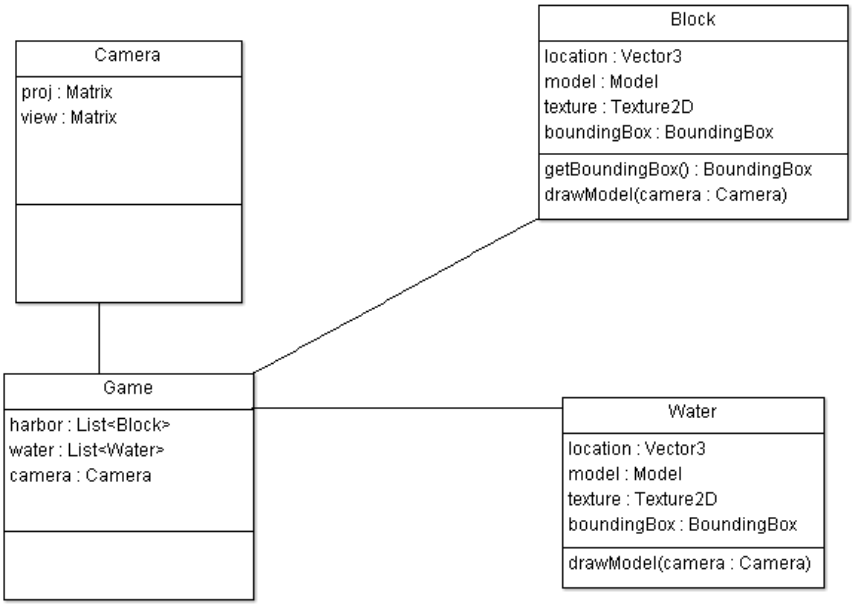
Peliin toteutettiin myös erilaisia esteitä pelaajan vaaraksi. Näiden tarkoitus on vaikeuttaa peliä tehden siitä haastavampaa pelaajalle:

1. Törmäystunnistus laiturin kanssa: Mikäli laiva osuu laituriin, se uppoaa ja pelaaja joutuu aloittamaan kentän alusta.
2. Veneet satamassa: Satamassa risteilee muita pienempiä veneitä, joita pelaajan täytyy väistää. Mikäli pelaaja törmää näihin veneisiin, joutuu hän aloittamaan kentän alusta.
3. Poijut: Satamassa on poijuja, joihin pelaaja ei saa törmätä. Poijut aiheuttavat pelaajan epäonnistumisen tehtävässä, jos hän upottaa kolme niistä. Tämän jälkeen pelaaja joutuu aloittamaan taas kentän alusta.

3.3 Pelimoottorin suunnittelu

Pelimoottoria suunnitellessa on otettava huomioon se, että tuleeko pelistä vuoropohjainen vai reaaliaikainen peli. Vuoropohjaisessa pelissä pelilogiikka ei laske tapahtumia vastakuin pelaaja on antanut oman syötteensä. Reaaliaikasessa pelissä pelitapahtumat kulkevat eteenpäin huolimatta siitä, onko pelaaja vaikuttanut pelaajahahmoonsa.

Harbor Madness on suunniteltu reaaliaikaiseksi peliksi, jolloin on hyvä tehdä jokaisesta pelaajasta, vihollisesta sekä pelialueen osasta jokaiselle oma olionsa, jotka tallennetaan listaan. Tällöin on helppo käydä jokainen niistä yksitellen läpi ja suorittaa jokaiselle oma toiminta. Pelialueen osien liitäntä pelirunkoon on esitelty kuviossa 1.



KUVIO 1. Luokkakaavio pelialueen olioista ja niiden liitoksista.

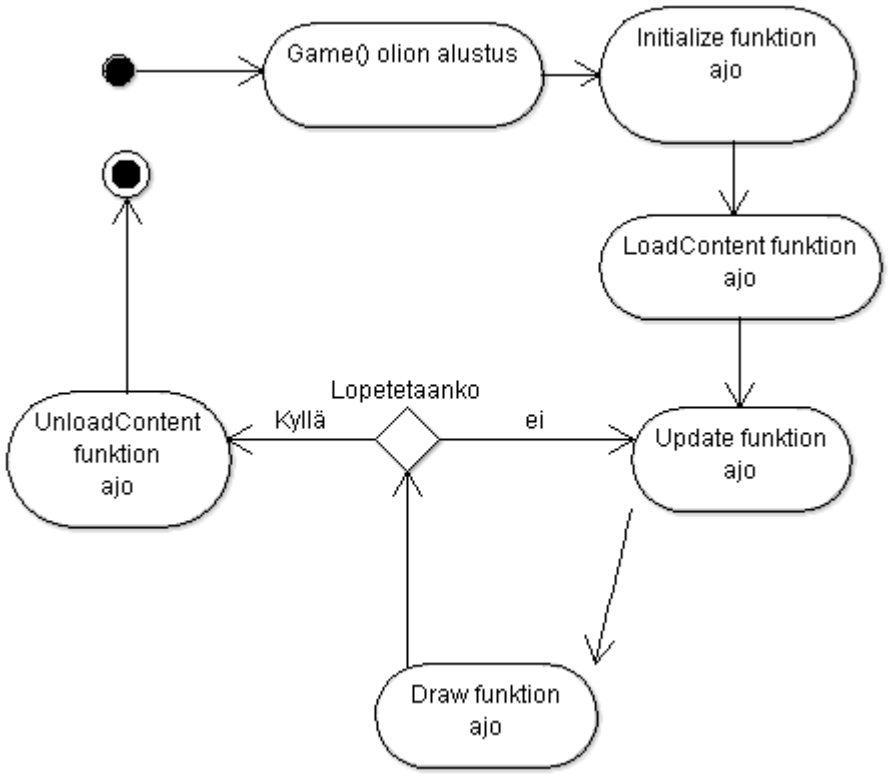
Jokainen pelin sisältämä objekti, pelaaja tai vihollinen, liikkuu reaaliaikaisesti, eli ei odota pelaajalta jotain tiettyä toimintoa tehdäkseen jotain. Tämän vuoksi pelimoottori täytyy suunnitella automaatioksi, jolle annetaan vain ulkopuolisia syötteitä, jonka mukaan se muuttaa käytöstään.

4 TOTEUTUS

XNAn parhaimpia puolia pelinkehityksessä on se, että itse pelilogiikan alla olevaa runkoa ei tarvitse itse tehdä vaan kehittäjä pystyy heti projektin luotuaan lähtemään kehittämään pelilogiikkaa.

4.1 XNA:n perustoiminnallisuus

Projektin valmispohjassa on valmiiksi tehty Game-luokan olio, joka hoitaa XNA:n moottorin perustoiminnallisuuden. Luokassa on valmiina 5 kappaletta vakio funktioita, joita tarvitaan pelin pyörittämiseeeen. Ohjelman perustoiminnallisuus on esitetty kuviossa 2.



KUVIO 2. Aktiviteettikaavio Game-luokan toiminnallisuudesta.

Initialize-metodi kutsutaan heti Game-olion luomisen jälkeen. Tähän funktioon on hyvä sijoittaa kaikki toiminta jota tarvitaan pelin luomiseen ja sen asetuksien säätämiseksi. Tässä metodissa on hyvä asettaa kaikki pelin vaatimat asetukset muistiin.

LoadContent-metodin kutsu suoritetaan Initialize-metodin jälkeen. Kaikki sovelluksen ulkopuolinen sisältö olisi hyvä ladata muistiin tässä funktiossa, sillä tätä funktiota kutsutaan vain kerran ohjelman alussa.

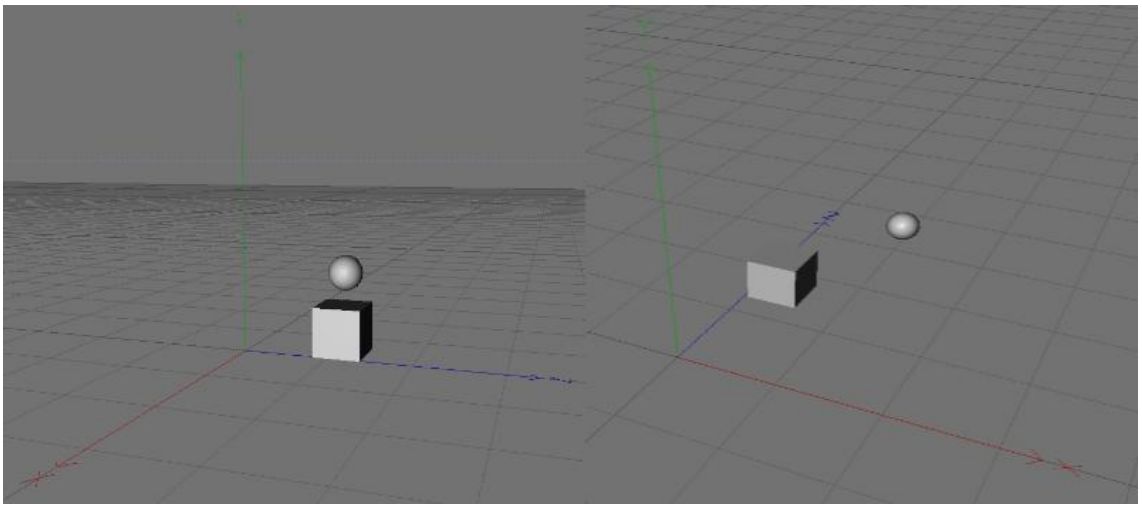
UnLoadContent-metodia kutsutaan kuten LoadContent-metodia, vain kerran ohjelman aikana. Tässä metodissa on tarkoitus suorittaa muistialueen tyhjennystä, vapauttaen kaikki graafinen sisältö muistista.

Update- ja Draw-metodit ovat pelin rungon pohja. Näitä kutsutaan pelin aikana tasaisin väliajoin. Update-metodiin sijoitetaan kaikki se toiminnallisuus, joka pyörittää peliä. Draw-metodissa suoritetaan sen sijaan kaikki grafiikkaan liittyvä toiminnallisuus.

4.2 3D-grafiikka

Jotta voidaan piirtää mallit ja sekä muut graafiset elementit ruudulle, kehittäjän täytyy ymmärtää kolmiulotteisuuden perusteet tietokonegrafiikassa. Paperille piirrettäessä on kaksi akselia, X- ja Y-akselit joiden avulla saadaan kohdistettua kuviot. 3D piirrossa tarvitaan vielä syvyyttä kuvaavan akselin nimeltään Z-akseli. Näillä kolmella akselilla saadaan sijoitettua objektit pelimaailmaan. Kuvassa 2 nähdään kaksi objektia 3D-mailmassa kahdesta eri kuvakulmasta. Kuvassa nähdään myös XYZ-akselit. Molemmissa kuvissa objektit ovat maailmaan nähden samassa paikassa, mutta katselukulman muutos aiheuttaa objektien näennäisen siirtymisen.

XNA:ssa käsitellään paikkaa ja suuntaa vektoreina. Vektori on suure, joka sisältää kolmen eri akselin arvot. Näiden avulla voidaan myös laskea liikkumiseen tarvittavia laskuja helposti.



KUVA 2. Kaksi mallia 3D-mailmassa.

4.3 Syötteet

Syötteiden lukeminen pelissä tarkoittaa sitä tapaa jolla tarkastellaan pelaajalta tulevaa informaatiota. Ellei pelaajalta pystytä saamaan syötteitä, ei peliä voi kutsua interaktiiviseksi kokemukseksi.

XNA:ssa syötteen lukeminen hoidetaan jokaiselle syötelaitteelle tehdyn olion avulla. Nämä 4 laitetta ovat seuraavat:

- GamePad: Tämä on joko Windowsin tai XBOX 360:n padia varten oleva olio
- Keyboard: Tämä on Windows koneiden näppäimistöä varten oleva olio
- Mouse: Tämä on Windowsin hiirtä varten oleva olio
- TouchPanel: Tämä on WP7 puhelinten kosketusnäyttöä varten oleva olio

Jokaisella näillä, pois lukien TouchPanel, on tietue, jonka avulla voidaan lukea laitteen näppäinten tai muiden syöttölaitteiden tilan. Listauksessa 1 esitetään koodi, jossa tarkistetaan näppäimistön nuolinäppäinten tilan.

```
KeyboardState keyb = Keyboard.GetState();
if (keyb.IsKeyDown(Keys.Up) && gameState == GAME)
    player.setPower(2);
if (keyb.IsKeyDown(Keys.Down) && gameState == GAME)
    player.setPower(-2);
if (keyb.IsKeyDown(Keys.Left) && gameState == GAME)
    player.setRudder(4.0f);
if (keyb.IsKeyDown(Keys.Right) && gameState == GAME)
    player.setRudder(-4.0f);
if (keyb.IsKeyUp(Keys.Right) && keyb.IsKeyUp(Keys.Left) &&
gameState == GAME)
    player.setRudder(0f);
if (keyb.IsKeyUp(Keys.Down) && keyb.IsKeyUp(Keys.Up) &&
gameState == GAME)
    player.setPower(0f);
```

LISTAUS 1. Näppäimistön tilan tarkistaminen nuolinäppäinten osalta.

Ensimmäisenä otetaan näppäimistön tilan talteen muuttujaan keyb. Tämän jälkeen käydään jokainen näppäin erikseen läpi, tarkistaen onko sen tila painettuna. Mikäli se on painettuna, annetaan pelaaja-oliolle komentoja sen mukaan, mitä halutaan kyseisestä näppäimestä tapahtuvan. Tässä tapauksessa nuolinäppäimet käskyttävät laivan moottoreita ja ruoria. Tämän lisäksi tarkistetaan myös IsKeyUp-metodin avulla sen, pitääkö laivalle antaa myös tieto siitä milloin moottoria ei käytetä.

Hiiren ja gamepad:n käsittely tapahtuu periaatteessa samalla tavalla kuin näppäimistön käsittely. Kummallakin oliolla on metodi nimeltä GetState, jolla saadaan haettua hiiren ja gamepadin sen hetkinen tila.

Gamepadin ja hiiren näppäinten painallukset luetaan samalla lailla kuin näppäimistön, tosin gamepadille tarvitaan yksi muutos. Sen GetState-metodille annetaan pelaajanumero, jolla erotellaan maksimissaan 4 gamepadia. Hiirellä ja gamepadilla on myös analogiset syötteet. Hiiren liikkuminen, sekä gamepadin analogiset sauva-ohjaimet.

Gamepadin GetState-metodin arvoista hakemalla saadaan sauvaohjaimien asennot X ja Y koordinaatteina. Nämä arvot ovat lukemien -1 ja 1 väliltä, 0 ollessa kohta, jossa ohjaimet ovat lepotilassa.

Hiirellä saadaan otettua monta eri arvoa sen sijainnista ja liikkeestä. Hiiren X ja Y arvot kertovat osoittimen sijainnin kyseisen sovelluksen ikkunassa. Näiden lisäksi pystytään ottamaan molempien akseleiden muutosarvot myös suoraan hiiren GetState metodilta, jotka kertovat kuinka paljon hiiren osoitin on liikkunut viime mittaushetkeen verrattuna. Tätä muutosarvoa käytetään yleensä peleissä, jossa kamera on sama kuin pelaajan näkemä kuva, jolloin hiiren liike vaikuttaa suoraan katseen suuntaan.

Tämä kaikki on hyvä tehdä XNA:n update-metodin sisällä, jota kutsutaan pelin aikana tietyn väliajoin. Tällöin saadaan laskettua ohjelman sisäisen ajan kanssa haluttu muutosnopeus, ettei syötteiden antonopeus kasva liian suureksi, vaikeuttaen täten käyttöliittymän käyttämistä.

4.4 Kamera

3D-kamera toimii samalla tavalla kuin oikeakin kamera. Sillä on sijainti, kuvaussuunta ja linssinominaisuudet. Näiden tietojen pohjalta renderöintiä varten lasketaan kaksi matriisia, Projectio-, sekä View-matriisi.

Ensin määritetään kameran suunta, eli View-matriisin, sen sijainnin ja katseen kohteen avulla. Tämän jälkeen näytön kuvasuhteen, katsekuvakulman ja piirtoetäisyyksien avulla saadaan laskettua projektio-matriisin. Näiden laskeminen on esitetty listauksessa 2.

```
GraphicsDeviceManager graphics = new
GraphicsDeviceManager(this);
Matrix view = Matrix.CreateLookAt(location, s, Vector3.Up);
Viewport viewport = g.GraphicsDevice.Viewport;
float aspectRatio = (float)viewport.Width /
(float)viewport.Height;
Matrix proj= Matrix.CreatePerspectiveFieldOfView(viewAngle,
aspectRatio, nearClip, farClip);
```

LISTAUS 2. Kameran laskenta renderöintiä varten.

Projectio-matriisi on kuvaus alueesta, joka otetaan renderöintiin mukaan. Siinä määritellään kolmiuloitteinen alue, joka otetaan renderöinnissä huomioon. Kaikki tämän alueen ulkopuolelle jäävät osat ja alueet jätetään piirtämättä.

4.5 3D-mallit

Malli siis koostuu joko yhdestä, tai useasta osasta nimeltään mesh. Nämä meshit ovat yksittäisiä mallin osia, joilla on kaikilla oma muotonsa, sekä omat tekstuurinsa. Jokainen meshi koostuu myös pienemmistä osista. Näitä osia kutsutaan mesh part-nimellä.

Pelin kaikki mallit on luotu ennalta jo Cinema4D ohjelmalla. Jotta nämä mallien osat saadaan käyttöön, tulee ne ladata Content-loaderin avulla mallin tietueisiin. Modelin lataus on esitetty listauksessa 3.

```
Model block;
Content.Load<Model>(”Block”);
```

LISTAUS 3. Modelin lataaminen muistiin Content-olion avulla.

Jokaiselle mallille luodaan oma olio, joka hoitaa mallin sijainnin pelialueella, sekä kaikki tarvittavat tiedot. Listauksessa 4 on esitetty satamalaiturin palan tiedot.

```
public Block(Model m, Texture2D t, Vector3 l)
{
    model = m;
    texture = t;
    location = l;
    boundingBox = new BoundingBox(location - new
       Vector3(25.0f, 10.0f, 25.0f), location + new
       Vector3(25.0f, 10.0f, 25.0f));
}
```

LISTAUS 4. Satama-olion alustus-funktio.

Olioon tallennetaan modeli, tekstuuri, sijainti sekä muodostetaan törmäystunnistuksessa tarvittava rajausmalli tietue, nimeltään BoundingBox.

Modelin renderöimisessä tarvitaan kameran avulla muodostettuja matriiseja, joiden avulla XNA tietää kuuluuko se piirtää ruudulle vai jääkö se kuvan ulkopuolelle. Listauksessa 5 esitetään modelin renderöiminen.

```
public void drawModel(Camera c)
{
    foreach (ModelMesh mesh in model.Meshes)
    {
        foreach (BasicEffect e in mesh.Effects)
        {
            e.Projection = c.proj;
            e.View = c.view;
            e.World = Matrix.CreateTranslation(location);
            e.Texture = texture;
            e.TextureEnabled = true;
            e.LightingEnabled = true;
            e.DirectionalLight0.Enabled = true;
            e.DirectionalLight0.Direction =
              Vector3.Normalize(new Vector3(10,-10,10));
            e.DirectionalLight0.DiffuseColor =
              Color.White.ToVector3();
            e.DirectionalLight1.Enabled = true;
            e.DirectionalLight1.Direction =
              Vector3.Normalize(new Vector3(-10,-10,-10));
            e.DirectionalLight1.DiffuseColor =
              Color.White.ToVector3();
            e.DirectionalLight2.Enabled = true;
            e.DirectionalLight2.Direction =
              Vector3.Normalize(new Vector3(0, -10, 0));
            e.DirectionalLight2.DiffuseColor =
              Color.White.ToVector3();
        }
        mesh.Draw();
    }
}
```

LISTAUS 5. Modelin renderöinti-funktio.

Malli renderöidään meshi kerrallaan, jolloin voidaan asettaa niille tarvittaessa esim. eri valaistuksen. Jokaisen meshin efektiin asetetaan kameralta saadut matriisit, jotka määrittävät sen paikan ruudulla modelin sijaintimatriisin avulla, joka lasketaan modelin paikkavektorista.

Sen jälkeen asetetaan tekstuuri meshille, sekä käynnistetään valaistuksen ja tekstuurin käsittely. Valon lähteitä perus efektille voidaan määritellä kahdeksan kappaletta. Tässä

esimerkissä on asettettu kolme kappaletta eri suunnista tulevia valoja, jotta koko malli olisi sopivasti valaistu ilman suurempia varjokohtia. Lopuksi käsketään meshiä piirtämään itsensä, kutsumalla sen draw-metodia.

4.6 Fysiikkalaskenta ja liikkuminen

Laivan liikkumista stimuloidaan pelissä suhteellisen alkeellisella fysiikanlaskennalla. Alkeellisuus tarkoittaa sitä, että pelissä lasketaan vain nopeudet ja hidastumiset laivalle. Fysiikkamoottori ei laske veden vastusta laivaa kohtaan muuten kuin kertoimen kautta.

Pelissä on ajateltu laivan liikkuminen jatkuvuuden lain, sekä kiihtyvyyksien kautta. Laivan nopeuden muutos saadaan jakamalla moottorien antama teho laivan massalla. Tämä muutosarvo lisätään laivan nykyiseen nopeuteen ja kerrotaan veden vastusarvolla, jolloin saadaan lopullinen nopeus sille hetkelle. Sen lisäksi nopeus on hyvä rajoittaa tiettyyn arvoon asti, jotta peli pysyy mielekkäänä. Laskenta ja nopeuden rajoitus esitetään listauksessa 6.

```
if (Math.Abs(enginePower) > Math.Abs(power))
    enginePower = power;
speed += enginePower / mass;

speed *= 1.0f - waterResistance;

if (Math.Abs(speed) > MAXSPEED)
{
    if (speed > 0)
        speed = MAXSPEED;
    else
        speed = -MAXSPEED;
}
```

LISTAUS 6. Pelaajan aluksen nopeuden laskenta.

Nopeuden laskemisen jälkeen voidaan laskea laivalle uusi sijainti pelimaailmaan. Sijainnin muutos lasketaan vektoreiden avulla. Ensimmäisenä laivalle täytyy laskea suunta ja asento, ennen kuin voimme siirtää sen oikeaan paikkaan. Jotta laiva ei kääntyisi paikallaan, otetaan huomioon sen nopeus kääntymisessä. Mikäli nopeus on muuta kuin nolla, kerrotaan sen ja maksiminopeuden välinen suhde peräsimen arvoon,

jolloin saamme nopeuden mukaan skaalautuvan kääntymisen asteluvun. Tämän avulla laskemme Quaternion-struktuurin laivan Y-akselin ympärillä käännettynä. Tämä toiminta esitetään listauksessa 7.

```
if (speed != 0)
{
    float turn = s_rudder * (speed / MAXSPEED);
    Quaternion rot = Quaternion.CreateFromAxisAngle(
         Vector3.Up, MathHelper.ToRadians(turn));
    shipHeading *= rot;
}

Vector3 v = new Vector3(speed, 0.0f, 0.0f);
Matrix forward = Matrix.CreateFromQuaternion(shipHeading);
v = Vector3.Transform(v, forward);

location.Z += v.Z;
location.X += v.X;
```

LISTAUS 7. Pelaajan laivan sijainnin laskenta.

Quaternion-struktuurin avulla saadaan sijoitettua laiva oikeaan paikkaan pelimaailmassa. Siirtoon tarvitaan vektoreita. Ensin muodostamme 3D-vektorin, johon annetaan X-akselin arvolle laivan nopeuden arvo. Muut arvot jätetään nollaksi, koska laivan paikkaa halutaan siirtää vain yhteen suuntaan.

Seuraavaksi vektoria käännetään aikaisemmin lasketusta suunnasta muodostetun matriisin avulla. Tämä tapahtuu kutsumalla Vector3 olion Transform-funktiota, joka laskee etäisyysvektorin ja suuntamatriisin avulla uuden vektorin. Tämä uusi vektori sisältää arvot, jotka lisäämällä laivan nykyiseen sijaintiin, muodostaa se laivalle uuden sijainnin.

Modelin asemoimisessa pitää ottaa huomioon, että modelin asentoa laskettaessa käytetään laskennallisena referenssipisteenä 3D-maailman keskipistettä, eli origoa, sekä maailman akseleita. Tämän vuoksi modeli täytyy kääntää oikeaan asentoon, ennen kuin se viedään haluttuun paikkaan maailmassa. Jos modeli on siirretty ensin lopulliseen sijoituspaikkaansa ja sen jälkeen pyöritetään, muuttaa pyöritys modelin paikkaa, koska pyörittämisen referenssipisteenä käytetään maailman origoa, eikä modelin keskipistettä.

4.7 Törmäykset

Törmäyksentunnistus peleissä on erittäin tärkeätä. Ilman sitä ei pystyttäisi tietämään, milloin pelaaja saavuttaa kohteen tai milloin pelaaja on törmännyt johonkin. Ilman törmäystunnistusta ei voi tehdä kunnollista peliä, jossa tarvitaan paikkaan ja aikaan liittyvää reagointia. Törmäystunnistus on sinällään helppo toteuttaa, mutta reagoiminen oikealla tavalla törmäykseen on pelin kannalta tärkein asia.

Törmäystunnistus XNA:ssa käsitellään rajausmallien BoundingBox tai BoundingSphere avulla. Nämä rajausmallit saadaan kutsuttua mallista suoraan metodin avulla tai määrittää itse. Nämä rajausmallit sisältävät metodin, jolla voidaan tarkistaa leikkaako kaksi rajausmallia keskenään. Listauksessa 8 esitetään koodi, jolla tarkistetaan osuuko pelin laiva laituriin.

```
foreach (Block b in harbor)
{
if (bs.Intersects(b.boundingBox) && playerAlive && !noClip)
{
    player.fullStop();
    if (gameState == GAME)
        playerAlive = false;
    }
}
```

LISTAUS 8. Törmäyksen tunnistus pelaajan ja sataman välillä.

Koodi käy jokaisen satamanosan rajausmallin läpi kerrallaan ja tarkistaa törmääkö se laivan rajausmalliin. Jos osuma on tullut ja pelaaja on elossa, eikä törmäyksenesto ole pois päältä, pelaajan alus pysäytetään ja asetetaan pelaajan status kuolleeksi.

Jokainen pelin mahdollinen törmäys tunnistetaan samalla tavalla. Pelaajan laivan törmäystä tarkistetaan sataman osia, soutuvenettä, poijuja ja maalia vastaan. Riippuen törmäyksen kohteesta, pelissä suoritetaan asianmukainen toiminta.

5 VIIMEISTELY

Tässä kappaleessa käsitellään pelin elävöittämiseen tarkoitettuja osioita XNA:sta. Nämä ovat äänet sekä grafiikka. Grafiikka jakautuu tässä kappaleessa käsittelemään Tekstuureita sekä High Level Shading Language -kieltä, eli grafiikkakortilla olevien shadereiden ohjelmointia.

5.1 Äänet

Äänet ovat pelin tunnelman luojia. Ilman mitään musiikkia tai ääniefektejä, pelikokemuksesta tulee yleensä tasapaksu ja tunteita herättämätön kokemus.

XNA tukee MP3, WAV ja WMA äänitiedostoja, jotka luetaan SoundEffect oliolle käyttäen apuna Content olion Load-funktiota. Tämän jälkeen äänitiedosto on toistettavissa suoraan, mutta sitä ei suositella tehtävän näin.

Ääniefektistä tehdään sen jälkeen instanssi, jolla kontrolloidaan sen soittoa pelin aikana. Tämän lisäksi tarvitaan vielä AudioEmitter ja AudioListener luokkien muuttujat, joilla määritellään äänen lähteen ja kuuntelijan paikat.

AudioListenerin, eli kuuntelija-olion, paikka on yleensä peleissä joko kamera, tai pelihahmo. Täten voidaan asettaa AudioListenerin paikkakoordinaatit samaksi kuin pelaajan, jolloin pelikokemus siirtyy enemmän pelin sisälle.

AudioEmitterin paikkakoordinaatit asetetaan kunkin äänilähteen luokse. Jokaisella pelihahmolla on yleensä yksi lähdekoordinaatti äänelle, jota käytetään äänen muodostamisen yhteydessä.

Nämä kaksi, lähde ja kuuntelija, annetaan seuraavaksi äänen instanssille tiedoksi, jolloin XNA osaa laskea äänen toistamisen arvot kohdalleen, jotta kuullaan ääni tulevaksi sieltä, mistä se pelissä tulee verraten kuuntelijan asemaan. Listauksessa 9 esitetään yhden äänen lataus, asetus ja toisto.

```
soundEngine = Content.Load<SoundEffect>("engine_2");
soundEngineInstance = soundEngine.CreateInstance();
soundHyperspaceActivation =
       Content.Load<SoundEffect>("Engine");

if (keyb.IsKeyDown(Keys.Up) && gameState == GAME)
{
if (soundEngineInstance.State == SoundState.Stopped)
{
    soundEngineInstance.Volume = 0.75f;
    soundEngineInstance.IsLooped = true;
    soundEngineInstance.Play();
}
else
    soundEngineInstance.Pause();
}
```

LISTAUS 9. Äänilähteen muodostaminen ja toistaminen.

5.2 Tekstuurit

Tekstuurilla 3D-grafiikassa tarkoitetaan modelin päälle pinnoitettua kuvaa. Tekstuurit tallennetaan erilliseen bittikarttatiedostoon, josta se ladataan ja asetetaan modelin päälle.

Jokainen modelin verteksi saa kohdistusarvon, jolla osoitetaan paikka tekstuurista. Näiden kohdistusarvojen avulla XNA laskee automaattisesti sen, kuinka tekstuuri piirretään 3D-modelin päälle.

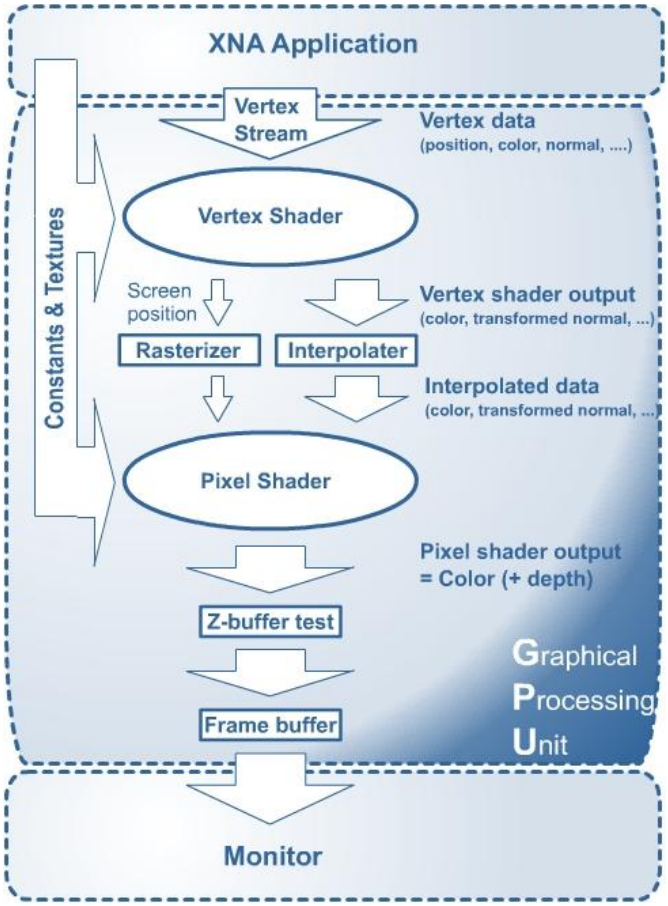
Tekstuurin kohdistusarvot ovat aina nollan ja yhden väliltä. XNA laskee tämän suhdeluvun avulla bittikarttakuvasta oikean kohdan, joka piirretään verteksin kohdalle ja niiden välille. Bittikartan vasen yläkulma on origo, eli nollapiste. Koordinaattiluvut kasvavat lineaarisesti kohti oikeaa alakulmaa, jossa koordinaattiarvot saavuttavat luvun yksi.

Tekstuurien sijasta voidaan myös käyttää liukuvärejä, jolloin jokaiselle modelin verteksille määritellään väriarvo. Tällöin verteksien välille saadaan tasainen liukuväriskaalan eri verteksien välille.

5.3 HLSL

HLSL, eli High Level Shading Language, on kieli, jolla voidaan ohjelmoida suoraan näytönohjaimen grafiikkasuoritinta. HLSL on C-kielen tyylinen korkean tason kieli, johon on lisätty ominaisuuksia ja toimintoja grafiikkasuoritinta ajatellen. Jotta voidaan ymmärtää miten HLSL toimii, ensin pitää ymmärtää grafiikkasuorittimen shader-putken toiminta.

XNA-ohjelma syöttää verteksien tiedot ensin grafiikkasuorittimen vertex-shader nimiselle lohkolle, joka annettujen parametrien ja tietojen mukaan käsittelee jokaisen verteksin erikseen. Tässä lohkossa verteksien ja tietojen perusteella lasketaan jokaisen pikselin paikka kuvaruudulla 3D-maailman sijainnin perusteella.



KUVA 2. Grafiikkasuorittimen kaaviokuva kuvanmuodostuksen osalta (http://www.riemers.net/images/Tutorials/XNA/Csharp/Series3/XNA%20Tutorial%202%20-%20HLSL%20introduction.jpg)

Tämän jälkeen tieto viedään rasterin ja iterpolaattorin läpi pixel-shaderille, jossa käsitellään jokainen pikseli, eli kuvapiste, erikseen. Pikseleitä voidaan tässä osiossa vielä muokata sijainnin, värin ja läpinäkyvyyden osalta helposti.

Tämän jälkeen kaikki tieto kuvasta viedään Z-bufferi testin läpi, joka laskee vielä etäisyyden perusteella mikä pikseli piirretään minkäkin päälle, jotta lähin jää näkyviin ja kauimmaiset kohteet ovat takana. Kun kaikki laskutoimitukset ja tarkistukset on tehty, viedään kuva lopulta monitorin ruudulle. Tämä toiminnallisuus esitetään kuvassa 2.

Itse pelin tekemiseen ja sen toimintaan HLSL ei vaikuta, vaan sitä käytetään parantamaan itse kuvaa ruudulla, jonka peli muodostaa. Koska jokainen verteksi menee vertex-shaderin läpi ja jokainen pikseli pixel-shaderin läpi ja näitä pystytään HLSL:n avulla ohjelmoimaan, voidaan tehdä niiden tiedoille mitä tahansa.

Tätä ei ole pakko käyttää pelin tekemiseen ja sen voi unohtaa kokonaan, mikäli on tyytyväinen XNA:n omiin, valmiisiin grafiikkaratkaisuihin. Mutta jos peliin halutaan lisätä vielä viimeinen silaus, on HLSL:n opiskelu suositeltavaa sen tehokkuuden vuoksi.

6 POHDINTA

XNA:n kaltaiset alustat ovat nykypäivänä yleistyneet hyvää tahtia, esimerkkinä myös Unity kehitystyökalu. Nämä työkalut antavat pelinkehittäjille hyvät eväänt tehdä sellaisia pelejä joita he itse haluavat. Tämä on mahdollistettu sillä, että kehitystyökalut hoitavat pelin vaatiman alustakohtaisen ohjelmoinnin vapauttaen näin kaiken kehityksen suoraan pelille, sekä mahdollistaen nopeat käännökset eri alustoille.

Haasteita tässä työssä oli opetella vektorilaskennan vaikutus 3D-ympäristöön, sekä matriisien käyttö mallien pyörittämiseen.

Työn tavoitteet saatiin täytettyä hyvin. Työssä saatiin tehtyä peli, joka voidaan portata alustalta toiselle vain hieman pelaajan syötteiden lukua koskevissa metodeissa. Peliä ei kuitenkaan päästy testaamaan kuin Windows ympäristössä, koska sopivia alustoja ei ollut saatavilla kehityksen aikana.

Tämä työ ei kuitenkaan käsittele kaikkea mitä XNA pystyy tarjoamaan. Puuttumaan jäi verkon yli toimiva osuus, joka mahdollistaa moninpelin toisen pelaajan kanssa, sekä HLSL jäi hieman suppeaksi, koska sitä ei tarvitse välttämättä toimivan pelin tekemiseen. Sekä Microsoft Liven tarjoamat hahmot ja sosiaalinen ympäristö jäi kokonaan käsittelemättä, koska sillä ei kuitenkaan ole vaikutusta tässä työssä olevaan peliin.

Nämä puuttuvat osa-alueet ovatkin seuraavana haasteena, jotta saadaan hyödynnettyä koko XNA:n potentiaali varteen otettavana pelinkehitys työkaluna.

LÄHTEET

Miller, T. & Johnson. D. 2011 XNA Game Studio 4.0 Programming. Boston: Pearson Education, Inc.

Microsoft XNA. Tulostettu 6.3.2012. http://en.wikipedia.org/wiki/Microsoft_XNA

LIITTEET

Sovelluksen lähdekoodi:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace HarborMadness
{
   class Levels
   {
      private List<string> levels;


      private static string level0 =
                  "111111111111" +
                  "100000000001" +
                  "102000008001" +
                  "100600060001" +
                  "100000000001" +
                  "100000600001" +
                  "106000000001" +
                  "100060006001" +
                  "108000000001" +
                  "100000000901" +
                  "100000000001" +
                  "111111111111";

      private static string level1 =
                  "111111111111" +
                  "100000000001" +
                  "102000000001" +
                  "100000000001" +
                  "100001100001" +
                  "100011110001" +
                  "100011110001" +
                  "100001100001" +
                  "100000000001" +
                  "100000000901" +
                  "100000000001" +
                  "111111111111";

      private static string level2 =
                  "111111111111" +
                  "100000000001" +
                  "102000000001" +
```

```
            "100000000001" +
            "100001100001" +
            "111111110001" +
            "111111110001" +
            "100001100001" +
            "100000000001" +
            "109000000001" +
            "100000000001" +
            "111111111111";

private static string level3 =
            "111111111111" +
            "100001000001" +
            "102001080901" +
            "100001000001" +
            "100001011111" +
            "100001000001" +
            "100001000001" +
            "100001000001" +
            "100001000001" +
            "100000000001" +
            "100000000001" +
            "111111111111";

private static string level4 =
            "111111111111" +
            "100010000001" +
            "109010000001" +
            "100011011001" +
            "110110001001" +
            "100010001001" +
            "100000001001" +
            "100010001001" +
            "111111111001" +
            "100000000001" +
            "105000000001" +
            "111111111111";

private static string level5 =
            "111111111111" +
            "191100000001" +
            "101000110401" +
            "101001110001" +
            "101011111111" +
            "101011100011" +
            "101011000001" +
            "101000001001" +
            "101100011001" +
            "100111110001" +
            "100000000001" +
            "111111111111";
```

```
private static string level6 =
            "111111111111" +
            "111111111111" +
            "111100000301" +
            "100001111111" +
            "111100000111" +
            "119111110001" +
            "110110000111" +
            "100111110001" +
            "101111111001" +
            "100111110001" +
            "100000000001" +
            "111111111111";

private static string level7 =
            "111111111111" +
            "100000000001" +
            "100000000301" +
            "111111111101" +
            "119009009101" +
            "111111111101" +
            "100000000001" +
            "100011110001" +
            "101111111001" +
            "100111110001" +
            "100000000001" +
            "111111111111";

public Levels()
{
   levels = new List<string>();
   initLevels();
}

private void initLevels()
{
   levels.Add(level0);
   levels.Add(level1);
   levels.Add(level2);
   levels.Add(level3);
   levels.Add(level4);
   levels.Add(level5);
   levels.Add(level6);
   levels.Add(level7);
}

public string getLevel(int index)
{
   try
   {
      return levels.ElementAt(index);
   }
```

```
            catch (ArgumentOutOfRangeException iore)
            {
                return iore.ToString();
            }
        }

        internal int maxLevel()
        {
            return levels.Count() - 1;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace HarborMadness
{

    class PlayerShip
    {
        public Vector3 startLocation;
        public float startHeading;
        public Model model;
        public Texture2D texture;
        public Vector3 location;
        public float rudder;

        private SoundEffect sound;
        private SoundEffectInstance sei;


        private bool drawShip = true;

        private bool tiltdirX = false;
        private bool tiltdirZ = false;
        private static float MAXSPEED = 3;
        private static float MAXTURN = 4;
        private float s_rudder;
        private float power;

        private float heading;
        private float speed;
        private float enginePower;
        private float mass;
        private float tiltX;
        private float tiltZ;
        private float waterResistance;

        private Quaternion shipHeading;

        public string data()
        {
            string ret;
            ret = "head: " + heading.ToString() + "\nspeed: " + speed.ToString()
```

```
            + "\nenginePower: " + enginePower.ToString() + "\ns_rudder: " + s_rudder.ToString()
            + "\ntiltdir: " + tiltX.ToString()
            + "\nPosition: " + location.ToString();
    return ret;
}

public void setSoundEffect(SoundEffect se)
{
    sound = se;
    sei = sound.CreateInstance();
}

public PlayerShip()
{
    speed = enginePower = 0.0f;
    power = 0.0f;
    shipHeading = Quaternion.Identity;
    mass = 100f;
    waterResistance = 0.00f;
}

public void setRudder(float r)
{
    if (r <= MAXTURN && r >= -MAXTURN)
        rudder = r;
}

public float getRudder()
{
    return rudder;
}

public void reset()
{
    tiltZ = tiltX = 0.0f;
    heading = startHeading;
    location = startLocation;
    drawShip = true;
    speed = 0.0f;
    power = 0.0f;
    shipHeading = Quaternion.Identity;
    Quaternion rot = Quaternion.CreateFromAxisAngle(Vector3.Up, MathHelper.ToRadians(heading));
    shipHeading *= rot;
}

public void setPower(float p)
{
    if (p != 0)
        sei.Volume = 1.0f;
    else
        sei.Volume = 0.3f;
```

```
    if (p > -10.1f && p < 10.1f)
        power = p;
}

public void startSound()
{
    sei.Play();
    sei.Volume = 0.3f;
}

public void stopSound()
{
    sei.Stop();
}

public float getPower()
{
    return power;
}

private void move()
{
    Vector3 v = new Vector3(speed, 0.0f, 0.0f);
    Matrix forward = Matrix.CreateFromQuaternion(shipHeading);
    v = Vector3.Transform(v, forward);

    location.Z += v.Z;
    location.X += v.X;
}

public List<BoundingSphere> getBoundingSpheres()
{
    List<BoundingSphere> bs = new List<BoundingSphere>();
    Vector3 bslocation = location;

    bslocation += Vector3.Transform(new Vector3(48.0f, 0.0f, -9.0f), Matrix.CreateFromQuaternion(shipHeading));
    bs.Add(new BoundingSphere(bslocation, 1.0f));
    bslocation = location;

    bslocation += Vector3.Transform(new Vector3(48.0f, 0.0f, 9.0f), Matrix.CreateFromQuaternion(shipHeading));
    bs.Add(new BoundingSphere(bslocation, 1.0f));
    bslocation = location;

    bslocation += Vector3.Transform(new Vector3(-45.0f, 0.0f, -9.0f), Matrix.CreateFromQuaternion(shipHeading));
    bs.Add(new BoundingSphere(bslocation, 1.0f));
    bslocation = location;

    bslocation += Vector3.Transform(new Vector3(-45.0f, 0.0f, 9.0f), Matrix.CreateFromQuaternion(shipHeading));
    bs.Add(new BoundingSphere(bslocation, 1.0f));
    bslocation = location;

    bslocation += Vector3.Transform(new Vector3(53.4f, 0.0f, 0.0f), Matrix.CreateFromQuaternion(shipHeading));
```

```
        bs.Add(new BoundingSphere(bslocation, 1.0f));
        bslocation = location;

        for (float i = -35.0f; i < 40.0f; i += 2.0f)
        {
            bslocation += Vector3.Transform(new Vector3(i, 0.0f, 0.0f), Matrix.CreateFromQuaternion(shipHeading));
            bs.Add(new BoundingSphere(bslocation, 9.5f));
            bslocation = location;
        }


        //bs.Add(new BoundingSphere(location, 10.0f));

        return bs;
    }

    public void updateShip(GameTime gameTime, bool alive)
    {
        if (alive)
        {
            if (tiltdirX)
                tiltX -= 0.001f;
            if (!tiltdirX)
                tiltX += 0.001f;

            if (Math.Abs(tiltX) > 0.05)
                if (tiltdirX)
                    tiltdirX = false;
                else
                    tiltdirX = true;

            if (tiltdirZ)
                tiltZ -= 0.0005f;
            if (!tiltdirZ)
                tiltZ += 0.0005f;

            if (Math.Abs(tiltZ) > 0.03)
                if (tiltdirZ)
                    tiltdirZ = false;
                else
                    tiltdirZ = true;

            if (s_rudder > rudder)
                s_rudder -= 0.05f;
            if (s_rudder < rudder)
                s_rudder += 0.05f;

            enginePower += power / 60;

            if (Math.Abs(enginePower) > Math.Abs(power))
                enginePower = power;
            speed += enginePower / mass;
```

```
            speed *= 1.0f - waterResistance;

            if (Math.Abs(speed) > MAXSPEED)
            {
                if (speed > 0)
                    speed = MAXSPEED;
                else
                    speed = -MAXSPEED;
            }
        }
        if (!alive)
        {
            if (location.Y > -20.0f)
                tiltZ = 0.5f;
            else
                tiltZ = 0.0f;
            if (location.Y > -30.0f)
                location -= new Vector3(0.0f, 0.5f, 0.0f);
            else
                drawShip = false;
        }
        move();


        //TODO: Physics calculation to ship movement.
        //      Change rotation -> must be calculated from position of rudder and ships speed. (Cant turn on spot)...
        Quaternion s_tilt = Quaternion.CreateFromAxisAngle(Vector3.UnitX, MathHelper.ToRadians(tiltX));
        s_tilt *= Quaternion.CreateFromAxisAngle(Vector3.UnitZ, MathHelper.ToRadians(tiltZ));
        if (speed != 0)
        {
            float turn = s_rudder * (speed / MAXSPEED);
            Quaternion rot = Quaternion.CreateFromAxisAngle(Vector3.Up, MathHelper.ToRadians(turn));
            shipHeading *= rot;
        }
        shipHeading *= s_tilt;
        location += Vector3.Transform(Vector3.Zero, Matrix.CreateFromQuaternion(shipHeading));
    }

    public void drawModel(Camera c)
    {
        if (!drawShip)
            return;

        Matrix[] transforms = new Matrix[model.Bones.Count];
        model.CopyAbsoluteBoneTransformsTo(transforms);

        Matrix rot = Matrix.CreateFromQuaternion(shipHeading);
        Matrix pos = Matrix.CreateTranslation(location);

        foreach (ModelMesh mesh in model.Meshes)
        {
```

```
            foreach (BasicEffect e in mesh.Effects)
            {
               e.TextureEnabled = true;
               e.Projection = c.proj;
               e.View = c.view;
               e.World = transforms[mesh.ParentBone.Index] * rot * pos;
               e.LightingEnabled = true;
               e.DirectionalLight0.Enabled = true;
               e.DirectionalLight0.DiffuseColor = Color.LightGoldenrodYellow.ToVector3();
               e.DirectionalLight0.Direction = Vector3.Normalize(new Vector3(15,-10,15));
               e.DirectionalLight1.Enabled = true;
               e.DirectionalLight1.DiffuseColor = Color.White.ToVector3();
               e.DirectionalLight1.Direction = Vector3.Normalize(new Vector3(0, -10, 0));
            }
            mesh.Draw();
         }
      }

      internal float getSpeed()
      {
         return speed;
      }


      internal void fullStop()
      {
         s_rudder = 0.0f;
         speed = 0.0f;
      }
   }
}
```

```
using System;

namespace HarborMadness
{
#if WINDOWS || XBOX
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        static void Main(string[] args)
        {
            using (Game1 game = new Game1())
            {
                game.Run();
            }
        }
    }
#endif
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace HarborMadness
{
   class Water
   {
      public Vector3 location;
      public Model model;
      public Texture2D texture1;

      public Water()
      {

      }

      public Water(Model m, Texture2D t1, Vector3 l)
      {
         model = m;
         texture1 = t1;
         location = l;
      }

      public void drawModel(Camera c, Vector3 lightDirection)
      {
         foreach (ModelMesh mesh in model.Meshes)
         {
            foreach (BasicEffect e in mesh.Effects)
            {
               e.LightingEnabled = true;
               e.DirectionalLight0.Direction = lightDirection;
               e.DirectionalLight0.DiffuseColor = Color.LightYellow.ToVector3();
               e.Projection = c.proj;
               e.View = c.view;
               e.World = Matrix.CreateTranslation(location);
               e.Texture = texture1;
               e.TextureEnabled = true;
               e.LightingEnabled = true;
            }
            mesh.Draw();
         }
```

```
        }

    } // End of Class
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace HarborMadness
{
   class Block
   {
      public Vector3 location;
      public Model model;
      public Texture2D texture;
      public BoundingBox boundingBox;

      public string datastr = "";

      public Block()
      {
      }

      public Block(Model m, Texture2D t, Vector3 l)
      {
         model = m;
         texture = t;
         location = l;
         boundingBox = new BoundingBox(location - new Vector3(25.0f, 10.0f, 25.0f), location + new Vector3(25.0f, 10.0f, 25.0f));
         datastr = "New Created";
      }

public void drawModel(Camera c)
{
   foreach (ModelMesh mesh in model.Meshes)
   {
      foreach (BasicEffect e in mesh.Effects)
      {
         e.Projection = c.proj;
         e.View = c.view;
         e.World = Matrix.CreateTranslation(location);
         e.Texture = texture;
         e.TextureEnabled = true;
         e.LightingEnabled = true;
         e.DirectionalLight0.Enabled = true;
         e.DirectionalLight0.Direction = Vector3.Normalize(new Vector3(10, -10, 10));
```

```
            e.DirectionalLight0.DiffuseColor = Color.White.ToVector3();
            e.DirectionalLight1.Enabled = true;
            e.DirectionalLight1.Direction = Vector3.Normalize(new Vector3(-10, -10, -10));
            e.DirectionalLight1.DiffuseColor = Color.White.ToVector3();
            e.DirectionalLight2.Enabled = true;
            e.DirectionalLight2.Direction = Vector3.Normalize(new Vector3(0, -10, 0));
            e.DirectionalLight2.DiffuseColor = Color.White.ToVector3();
        }
        mesh.Draw();
    }
} // End of drawModel()

    } // End of Class
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace HarborMadness
{
   class Boat
   {
      public Model model;
      //public Texture2D texture;
      public Vector3 location;
      private Vector3 startLocation;
      private float startHeading;

      public float heading;

      private Quaternion BoatHeading;

      private static float SPEED = 0.2f;

      public Boat()
      {
         heading = 0.0f;
         BoatHeading = Quaternion.Identity;
      }

      public BoundingSphere getBoundingSphere()
      {
         //BoundingSphere tmp = model.Meshes[0].BoundingSphere;
         //tmp.Center += new Vector3(0.0f, 5.0f, 0.0f);

         //Vector3 bslocation = location;

         //bslocation += Vector3.Transform(new Vector3(48.0f, 0.0f, -9.0f), Matrix.CreateFromQuaternion(Quaternion.CreateFromAxisAngle(Vector3.Up, MathHelper.ToRadians(heading)));
         return new BoundingSphere(location, 6.0f);
      }

      public string data()
      {
         return "Boat: " + location.ToString() + "|" + heading.ToString();
      }
```

```
public void reset()
{
   location = startLocation;
   heading = startHeading;
   BoatHeading = Quaternion.Identity;
}

public void setLocation(Vector3 l, float h)
{
   startLocation = l + Vector3.UnitY;
   startHeading = h;
   heading = startHeading;
   location = startLocation;
}

public void move()
{
   Vector3 v = new Vector3(SPEED, 0.0f, 0.0f);
   BoatHeading *= Quaternion.CreateFromAxisAngle(Vector3.Up, MathHelper.ToRadians(heading));

   Matrix forward = Matrix.CreateFromQuaternion(BoatHeading);
   v = Vector3.Transform(v, forward);

   location.Z += v.Z;
   location.X += v.X;

}

public void drawModel(Camera camera)
{
   Matrix[] transforms = new Matrix[model.Bones.Count];
   model.CopyAbsoluteBoneTransformsTo(transforms);

   Matrix rot = Matrix.CreateFromQuaternion(BoatHeading);
   Matrix pos = Matrix.CreateTranslation(location);

   foreach (ModelMesh mesh in model.Meshes)
   {
      foreach (BasicEffect e in mesh.Effects)
      {
         e.Projection = camera.proj;
         e.EnableDefaultLighting();
         e.View = camera.view;
         e.World = transforms[mesh.ParentBone.Index] * rot * pos;
         e.TextureEnabled = true;
         e.LightingEnabled = true;
         e.DirectionalLight0.Enabled = true;
         e.DirectionalLight0.DiffuseColor = Color.LightGoldenrodYellow.ToVector3();
         e.DirectionalLight0.Direction = Vector3.Normalize(new Vector3(15, -10, 15));
      }
      mesh.Draw();
   }
```

```
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace HarborMadness
{
   class Buoy
   {



      public Model model;
      public Vector3 location;
      public Texture2D texture;

      public bool enabled;

      public Buoy(Model m, Vector3 l, Texture2D t)
      {
         model = m;
         location = l;
         texture = t;
         enabled = true;
      }

      public string data()
      {
         return "Buoy: " + location.ToString();
      }

      public void setLocation(Vector3 l)
      {
         location = l;
      }

      public BoundingSphere getBoundingSphere()
      {
         if (enabled)
            return new BoundingSphere(location, 8.0f);
         else
            return new BoundingSphere(new Vector3(0, -600, 0), 1);
      }
```

```
        public void drawModel(Camera camera)
        {
            if (enabled)
            {
                foreach (ModelMesh mesh in model.Meshes)
                {
                    foreach (BasicEffect e in mesh.Effects)
                    {
                        e.Projection = camera.proj;
                        e.LightingEnabled = true;
                        e.DirectionalLight0.Enabled = true;
                        e.DirectionalLight0.Direction = Vector3.Normalize(new Vector3(10, -10, 10));
                        e.DirectionalLight0.DiffuseColor = Color.LightCoral.ToVector3();
                        e.View = camera.view;
                        e.World = Matrix.CreateTranslation(location);
                        e.Texture = texture;
                        e.TextureEnabled = true;

                    }
                    mesh.Draw();
                }
            }
        }
    }
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace HarborMadness
{
   class Camera
   {
      public Vector3 location;
      public Vector3 reference = new Vector3(0, 0, 10);

      public float rotationY = 0f;
      public float rotationX = MathHelper.PiOver4;
      public float rotationZ = 0f;
      public float height = 150f;
      private float oldheight;
      public Matrix view;
      public Matrix proj;

      public float viewAngle = 0.5f;
      public float nearClip;
      public float farClip;

      private Quaternion crot;

      public Camera()
      {
         crot = Quaternion.Identity;
         nearClip = 1f;
         farClip = 10000f;
      }

      public void updateCamera(GraphicsDeviceManager g, Vector3 s, bool topcam)
      {
         location = s;

         Vector3 v;
         if (topcam)
         {
            v = new Vector3(1f, 0f, 0f);
            if (oldheight == 0f)
               oldheight = height;
            height = 600f;
```

```
            }
            else
            {
                v = new Vector3(400f, 0.0f, 0.0f);
                if (oldheight != 0f)
                    height = oldheight;
                oldheight = 0f;
            }

            Matrix forward = Matrix.CreateRotationY(MathHelper.ToRadians(rotationY));
            v = Vector3.Transform(v, forward);

            location.Z += v.Z;
            location.X += v.X;
            location.Y += height;

            view = Matrix.CreateLookAt(location, s, Vector3.Up);

            Viewport viewport = g.GraphicsDevice.Viewport;
            float aspectRatio = (float)viewport.Width / (float)viewport.Height;

            proj = Matrix.CreatePerspectiveFieldOfView(viewAngle, aspectRatio, nearClip, farClip);

        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;

namespace HarborMadness
{

    /// <summary>
    /// This is the main type for your game
    /// </summary>
    ///
    public class Game1 : Microsoft.Xna.Framework.Game
    {


        Effect effect;

        GraphicsDeviceManager graphics;
        GraphicsDevice device;
        SpriteBatch spriteBatch;
        SpriteFont font;
        SpriteFont font2;

        string output;

        private static string HELP = "Arrows: move ship" +
                                     "\nWASD: Move camera" +
                                     "\nSPACE: select" +
                                     "\nENTER: Camera mode" +
                                     "\nESC: quit" +
                                     "\nStop your ship on the target to complete level";

        private int currentLevel = 0;
        private PlayerShip player;
        private Camera camera;
        private Levels levels;

        private Model target;
        private List<Buoy> buoy;
        private Boat boat;

        private List<Block> harbor;
        private List<Water> water;
```

```
private Vector3 finish;
private int gameState;
private int levelSelect = 0;

private Texture2D background;
private Texture2D startbg;

//private VertexBuffer waterVertexBuffer;
VertexDeclaration waterVertexDeclaration;

private VertexPositionColorTexture[] vpnt;

private int rammedB;
private bool playerAlive = true;
private bool gameReady = false;
private bool boatKilled = false;

private bool topcam = false;

private static int MENU = 1;
private static int GAME = 2;
private static int PAUSE = 3;
private static int NEXT = 4;

private string SUNKEN;
private string BOATKILLED;

private bool noClip = false;

private Keys reset = Keys.Space;
private Keys iddqd = Keys.F10;

private Texture2D selector;

private string hitpoint = "";

private KeyboardState oldkey;
public Game1()
{
   graphics = new GraphicsDeviceManager(this);

   Content.RootDirectory = "Content";
}

private Vector3 pos(int x, int y)
{
   return new Vector3(x * 50.0f, 0.0f, y * 50.0f);
}
/// <summary>
/// Allows the game to perform any initialization it needs to before starting to run.
/// This is where it can query for any required services and load any non-graphic
/// related content.  Calling base.Initialize will enumerate through any components
```

```
/// and initialize them as well.
/// </summary>
protected override void Initialize()
{
    // TODO: Add your initialization logic here
    oldkey = Keyboard.GetState();
    SUNKEN = "Oh, you sank!\nPress " + reset.ToString() + " to reset";
    BOATKILLED = "Oh, you ruined it!\nPress " + reset.ToString() + " to reset";
    gameState = MENU;
    levels = new Levels();
    player = new PlayerShip();
    boat = new Boat();
    buoy = new List<Buoy>();
    //player.location = new Vector3(0.0f, 0.0f, 0.0f);

    camera = new Camera();
    //camera.location = player.location + new Vector3(0, 400, -400);

    water = new List<Water>();
    harbor = new List<Block>();

    finish = new Vector3();

    device = GraphicsDevice;
    // createLevel();
    rammedB = 0;
    vpnt = new VertexPositionColorTexture[4];

    SetUpWaterVertices();
    waterVertexDeclaration = new VertexDeclaration(VertexPositionTexture.VertexDeclaration.GetVertexElements());

    base.Initialize();
}

private void SetUpWaterVertices()
{
    float waterHeight = 0.0f;

    vpnt[0].Position = new Vector3(550, waterHeight, 0);
    vpnt[0].Color = Color.LightBlue;
    //vpnt[0].Normal = Vector3.UnitY;
    vpnt[0].TextureCoordinate = new Vector2(0, 0);

    vpnt[1].Position = new Vector3(550, waterHeight, 550);
    vpnt[1].Color = Color.LightBlue;
    //vpnt[1].Normal = Vector3.UnitY;
    vpnt[1].TextureCoordinate = new Vector2(1, 0);

    vpnt[2].Position = new Vector3(0, waterHeight, 0);
    vpnt[2].Color = Color.LightBlue;
    //vpnt[2].Normal = Vector3.UnitY;
    vpnt[2].TextureCoordinate = new Vector2(0, 1);
```

```
            vpnt[3].Position = new Vector3(0, waterHeight, 550);
            vpnt[3].Color = Color.LightBlue;
            //vpnt[3].Normal = Vector3.UnitY;
            vpnt[3].TextureCoordinate = new Vector2(1, 1);
        }

        public void createLevel()
        {
            int i = 0;
            int levelWidth = 12;

            try
            {
                foreach(char c in levels.getLevel(currentLevel))
                {

                    if (c == '1')
                        harbor.Add(new Block(Content.Load<Model>("BasicBlock"), Content.Load<Texture2D>("Concrete"), pos(i / levelWidth, i % levelWidth)));
                    //if (c != '1')
                    //    water.Add(new Water(Content.Load<Model>("WaterBlock"), Content.Load<Texture2D>("Water"), pos(i / levelWidth, i % levelWidth)));
                    if (c == '2' || c == '3' || c == '4' || c == '5' )
                    {
                        if (c == '2')
                        {
                            player.startHeading = 0.0f;
                            camera.rotationY = 180.0f;
                        }
                        if (c == '3')
                        {
                            player.startHeading = 90.0f;
                            camera.rotationY = 270.0f;
                        }
                        if (c == '4')
                        {
                            player.startHeading = 180.0f;
                            camera.rotationY = 0.0f;
                        }
                        if (c == '5')
                        {
                            player.startHeading = 270.0f;
                            camera.rotationY = 90.0f;
                        }
                        player.startLocation = pos(i / levelWidth, i % levelWidth);
                    }
                    if (c == '9')
                        finish = pos(i / levelWidth, i % levelWidth);
                    if (c == '8')
                        boat.setLocation(pos(i / levelWidth, i % levelWidth), 0.0f);
                    if (c == '6')
```

```
                buoy.Add(new Buoy(Content.Load<Model>("Buoy"), pos(i / levelWidth, i % levelWidth), Content.Load<Texture2D>("Metal")));
                i++;
            }

        }
        catch (NullReferenceException nre) { output = nre.ToString(); }
        player.reset();
        gameReady = true;
        player.startSound();
    }

    /// <summary>
    /// LoadContent will be called once per game and is the place to load
    /// all of your content.
    /// </summary>
    protected override void LoadContent()
    {
        // Create a new SpriteBatch, which can be used to draw textures.
        spriteBatch = new SpriteBatch(GraphicsDevice);
        font = Content.Load<SpriteFont>("Font");
        font2 = Content.Load<SpriteFont>("Font2");
        selector = Content.Load<Texture2D>("Selector2");
        background = Content.Load<Texture2D>("Background");
        startbg = Content.Load<Texture2D>("startimage");
        target = Content.Load<Model>("Target");
        player.model = Content.Load<Model>("Ship");
        player.texture = Content.Load<Texture2D>("Metal");
        player.setSoundEffect(Content.Load<SoundEffect>("Fire_loop"));
        boat.model = Content.Load<Model>("BBoat");
        //boat.texture = player.texture;
        effect = Content.Load<Effect>("Effect2");
        // TODO: use this.Content to load your game content here
    }

    /// <summary>
    /// UnloadContent will be called once per game and is the place to unload
    /// all content.
    /// </summary>
    protected override void UnloadContent()
    {
        // TODO: Unload any non ContentManager content here
    }

    /// <summary>
    /// Allows the game to run logic such as updating the world,
    /// checking for collisions, gathering input, and playing audio.
    /// </summary>
    /// <param name="gameTime">Provides a snapshot of timing values.</param>
    protected override void Update(GameTime gameTime)
    {
        updateByKeyboard();
```

```
if (gameState == GAME || gameState == PAUSE)
{

   if (gameState == GAME)
   {
      player.updateShip(gameTime, playerAlive);
      boat.move();

      try
      {

         foreach (BoundingSphere bs in player.getBoundingSpheres())
         {

            if (bs.Intersects(new BoundingSphere(finish, 25.0f)) && player.getSpeed() < 0.05f && playerAlive)
            {
               nextLevel();
            }

            foreach (Block b in harbor)
            {
               if (bs.Intersects(b.boundingBox) && playerAlive && !noClip)
               {
                  hitpoint = bs.Center.ToString() + " - " + b.location.ToString() + " | " + gameReady.ToString()
                           + " | " + gameState.ToString();
                  player.fullStop();
                  if (gameState == GAME)
                     playerAlive = false;
               }
            }

            if ( bs.Intersects( boat.getBoundingSphere() ) && playerAlive && !noClip )
            {
               player.fullStop();
               if (gameState == GAME)
                  boatKilled = true;
            }

            foreach (Buoy b in buoy)
            {
               if (bs.Intersects(b.getBoundingSphere()) && playerAlive && !noClip)
               {
                  b.enabled = false;
                  rammedB++;
               }
            }
            if (rammedB > 2)
            {
               player.fullStop();
               if (gameState == GAME)
                  playerAlive = false;
```

```
                }
            }

            foreach (Block b in harbor)
            {
                if (boat.getBoundingSphere().Intersects(b.boundingBox))
                    boat.heading += 180.0f;

            }
        }
        catch (NullReferenceException nre) { output = nre.ToString(); }
    }

    camera.updateCamera(graphics, player.location, topcam);

  }

  if (gameState == NEXT)
  {
    camera.updateCamera(graphics, player.location, topcam);
  }

  if (gameState == MENU)
  {

  }
  base.Update(gameTime);
}

void updateByKeyboard()
{
  KeyboardState keyb = Keyboard.GetState();

  if (gameState == MENU)
  {
    if (keyb.IsKeyDown(Keys.Space))
    {
      harbor.Clear();
      water.Clear();
      currentLevel = levelSelect;
      createLevel();
      gameState = GAME;
    }

    if (keyb.IsKeyDown(Keys.Right))
      if (!oldkey.IsKeyDown(Keys.Right))
        levelSelect++;

    if (keyb.IsKeyDown(Keys.Left))
      if (!oldkey.IsKeyDown(Keys.Left))
        levelSelect--;
```

```
    if (levelSelect < 0)
        levelSelect = levels.maxLevel();

    if (levelSelect > levels.maxLevel())
        levelSelect = 0;

    if (keyb.IsKeyDown(Keys.Escape))
        if (!oldkey.IsKeyDown(Keys.Escape))
            this.Exit();
}

if (gameState == GAME || gameState == PAUSE)
{
    if (keyb.IsKeyDown(Keys.Insert))
    {
        if (!oldkey.IsKeyDown(Keys.Insert))
            nextLevel();
    }
    if (keyb.IsKeyDown(Keys.Delete))
    {
        if (!oldkey.IsKeyDown(Keys.Delete))
            prevLevel();
    }
    if (keyb.IsKeyDown(Keys.Enter))
    {
        if (!oldkey.IsKeyDown(Keys.Enter))
        {
            if (topcam)
                topcam = false;
            else
                topcam = true;
        }
    }

    if (keyb.IsKeyDown(Keys.Q))
        camera.location.Y += 10.0f;
    if (keyb.IsKeyDown(Keys.E))
        camera.location.Y -= 10.0f;

    if (keyb.IsKeyDown(Keys.A))
        camera.rotationY += 0.5f;
    if (keyb.IsKeyDown(Keys.D))
        camera.rotationY -= 0.5f;
    if (keyb.IsKeyDown(Keys.W))
        camera.height -= 4.0f;
    if (keyb.IsKeyDown(Keys.S))
        camera.height += 4.0f;

    if (camera.height < 100)
        camera.height = 100f;
    if (camera.height > 600)
        camera.height = 600f;
```

```
    if (keyb.IsKeyUp(Keys.Right) && keyb.IsKeyUp(Keys.Left) && gameState == GAME)
        player.setRudder(0f);
    if (keyb.IsKeyUp(Keys.Down) && keyb.IsKeyUp(Keys.Up) && gameState == GAME)
        player.setPower(0f);

    if (keyb.IsKeyDown(Keys.Up) && gameState == GAME)
        player.setPower(2);
    if (keyb.IsKeyDown(Keys.Down) && gameState == GAME)
        player.setPower(-2);
    if (keyb.IsKeyDown(Keys.Left) && gameState == GAME)
        player.setRudder(4.0f);
    if (keyb.IsKeyDown(Keys.Right) && gameState == GAME)
        player.setRudder(-4.0f);

    if (keyb.IsKeyDown(Keys.Back) && gameState == GAME)
        player.fullStop();

    if (keyb.IsKeyDown(reset) && (!playerAlive || boatKilled))
    {
        playerAlive = true;
        boatKilled = false;
        player.reset();
        boat.reset();
        foreach (Buoy b in buoy)
            b.enabled = true;
        rammedB = 0;
    }
    if (keyb.IsKeyDown(iddqd))
        if (!oldkey.IsKeyDown(iddqd))
        {
            if (noClip)
                noClip = false;
            else
                noClip = true;
        }

    if (keyb.IsKeyDown(Keys.Escape))
        if (!oldkey.IsKeyDown(Keys.Escape))
            gameState = MENU;

    if (keyb.IsKeyDown(Keys.P))
        if (!oldkey.IsKeyDown(Keys.P))
        {
            if (gameState == GAME)
                gameState = PAUSE;
            else
                gameState = GAME;
        }
}

if (gameState == NEXT)
```

```
{
   if (keyb.IsKeyDown(Keys.Insert))
   {
      if (!oldkey.IsKeyDown(Keys.Insert))
         nextLevel();
   }
   if (keyb.IsKeyDown(Keys.Delete))
   {
      if (!oldkey.IsKeyDown(Keys.Delete))
         prevLevel();
   }

   if (keyb.IsKeyDown(Keys.Escape))
      if (!oldkey.IsKeyDown(Keys.Escape))
         gameState = MENU;

   if (keyb.IsKeyDown(reset))
   {
      gameState = GAME;
   }

   if (keyb.IsKeyDown(Keys.Enter))
   {
      if (!oldkey.IsKeyDown(Keys.Enter))
      {
         if (topcam)
            topcam = false;
         else
            topcam = true;
      }
   }

   if (keyb.IsKeyDown(Keys.Q))
      camera.location.Y += 10.0f;
   if (keyb.IsKeyDown(Keys.E))
      camera.location.Y -= 10.0f;

   if (keyb.IsKeyDown(Keys.A))
      camera.rotationY += 0.5f;
   if (keyb.IsKeyDown(Keys.D))
      camera.rotationY -= 0.5f;
   if (keyb.IsKeyDown(Keys.W))
      camera.height -= 4.0f;
   if (keyb.IsKeyDown(Keys.S))
      camera.height += 4.0f;

   if (camera.height < 100)
      camera.height = 100f;
   if (camera.height > 600)
      camera.height = 600f;
}
oldkey = keyb;
```

```
}

private void nextLevel()
{
    gameState = NEXT;
    playerAlive = true;
    boatKilled = false;
    player.location = new Vector3(1500, 1500, 1500);
    currentLevel++;
    if (currentLevel > levels.maxLevel())
        currentLevel = levels.maxLevel();
    harbor.Clear();
    water.Clear();
    createLevel();
    player.reset();
    player.startSound();
}

private void prevLevel()
{
    playerAlive = true;
    boatKilled = false;
    currentLevel--;
    if (currentLevel < 0)
        currentLevel = 0;
    harbor.Clear();
    water.Clear();
    player.reset();
    createLevel();
    player.startSound();
}

/// <summary>
/// This is called when the game should draw itself.
/// </summary>
/// <param name="gameTime">Provides a snapshot of timing values.</param>
protected override void Draw(GameTime gameTime)
{
    if (gameState == GAME || gameState == PAUSE || gameState == NEXT)
        drawGame();

    if (gameState == MENU)
        drawMenu();

    base.Draw(gameTime);
}

private void drawLevels()
{
    Point first = new Point(50, 50);

    for (int i = 0; i < levels.maxLevel() + 1; i++)
```

```
    {
        spriteBatch.DrawString(font, (i+1).ToString(), new Vector2(first.X + (i * 40), first.Y), Color.SandyBrown);
    }
}

private void drawMenu()
{
    Rectangle backgroundRectangle = new Rectangle(0, 0,
        graphics.GraphicsDevice.Viewport.Width,
        graphics.GraphicsDevice.Viewport.Height);

    GraphicsDevice.Clear(Color.Transparent);

    GraphicsDevice.SamplerStates[0] = SamplerState.LinearClamp;

    spriteBatch.Begin(SpriteSortMode.Immediate, BlendState.AlphaBlend);

    spriteBatch.Draw(startbg, backgroundRectangle, Color.White);
    spriteBatch.DrawString(font2, "Select level: ", new Vector2(50, 40), Color.SandyBrown);
    spriteBatch.Draw(selector, new Vector2(42, 60) + new Vector2(levelSelect * 40, 0), Color.White);

    spriteBatch.DrawString(font2, HELP, new Vector2(10, 350), Color.SandyBrown);
    drawLevels();
    spriteBatch.End();
}

private void drawGame()
{
    Rectangle backgroundRectangle = new Rectangle(0, 0,
        graphics.GraphicsDevice.Viewport.Width,
        graphics.GraphicsDevice.Viewport.Height);

    //Vector3 ld = Vector3.Normalize(new Vector3(0, -1, -1));

    GraphicsDevice.Clear(Color.Transparent);

    GraphicsDevice.SamplerStates[0] = SamplerState.LinearClamp;

    spriteBatch.Begin(SpriteSortMode.Immediate, BlendState.AlphaBlend);
    spriteBatch.Draw(background, backgroundRectangle ,Color.White);

    spriteBatch.End();

    GraphicsDevice.BlendState = BlendState.Opaque;
    GraphicsDevice.DepthStencilState = DepthStencilState.Default;
    GraphicsDevice.RasterizerState = RasterizerState.CullCounterClockwise;
    GraphicsDevice.SamplerStates[0] = SamplerState.LinearWrap;

    try
    {
        foreach (Block b in harbor)
        {
```

```
            b.drawModel(camera);
        }
    }
    catch (NullReferenceException nre) { output = nre.ToString(); }

    foreach (ModelMesh mesh in target.Meshes)
    {
        foreach (BasicEffect e in mesh.Effects)
        {
            e.Projection = camera.proj;
            e.View = camera.view;
            e.World = Matrix.CreateTranslation(finish);
            e.Texture = player.texture;
            e.TextureEnabled = true;
            e.EnableDefaultLighting();
        }
        mesh.Draw();
    }

    foreach(Buoy b in buoy)
        b.drawModel(camera);

    boat.drawModel(camera);
    player.drawModel(camera);
    DrawWater();

    GraphicsDevice.SamplerStates[0] = SamplerState.LinearClamp;

    if (output != "DAS FINALE")
        output = boat.data();
    spriteBatch.Begin(SpriteSortMode.BackToFront, BlendState.AlphaBlend);
    //spriteBatch.DrawString(font2, output, new Vector2(0.0f, graphics.GraphicsDevice.Viewport.Height - 150), Color.Yellow);
    spriteBatch.DrawString(font2, hitpoint, new Vector2(0.0f, graphics.GraphicsDevice.Viewport.Height - 20), Color.Yellow);
    spriteBatch.DrawString(font, "Level: " + (currentLevel+1).ToString(), Vector2.One, Color.Red);
    if (noClip)
        spriteBatch.DrawString(font, "NoClip mode ON", new Vector2(0.0f,40.0f), Color.Red);
    if (!playerAlive)
    {
        spriteBatch.DrawString(font, SUNKEN, new Vector2(graphics.GraphicsDevice.Viewport.Bounds.Center.X,
graphics.GraphicsDevice.Viewport.Bounds.Center.Y) - font.MeasureString(SUNKEN) / 2, Color.Red);
    }
    if (boatKilled)
    {
        spriteBatch.DrawString(font, BOATKILLED, new Vector2(graphics.GraphicsDevice.Viewport.Bounds.Center.X,
graphics.GraphicsDevice.Viewport.Bounds.Center.Y) - font.MeasureString(SUNKEN) / 2, Color.Red);
    }
    if (gameState == PAUSE)
        spriteBatch.DrawString(font, "GAME PAUSED", new Vector2(graphics.GraphicsDevice.Viewport.Bounds.Center.X,
graphics.GraphicsDevice.Viewport.Bounds.Center.Y) - font.MeasureString("Game Paused") / 2, Color.Red);
    if (gameState == NEXT)
```

```
            spriteBatch.DrawString(font, "PRESS SPACE TO START", new Vec-
tor2(graphics.GraphicsDevice.Viewport.Bounds.Center.X, graphics.GraphicsDevice.Viewport.Bounds.Center.Y) -
font.MeasureString("PRESS SPACE TO START") / 2, Color.Red);
            spriteBatch.End();
        }

        private void DrawWater()
        {
            Vector3 ld = Vector3.Normalize(new Vector3(0, -1, -1));

            effect.CurrentTechnique = effect.Techniques["Technique1"];
            Matrix worldMatrix = Matrix.Identity;
            effect.Parameters["World"].SetValue(worldMatrix);
            effect.Parameters["View"].SetValue(camera.view);
            effect.Parameters["Projection"].SetValue(camera.proj);
            effect.Parameters["xTexture"].SetValue(Content.Load<Texture2D>("Water"));
            //effect.Parameters["xLightDirection"].SetValue(ld);

            effect.CurrentTechnique.Passes[0].Apply();

            //device.SetVertexBuffer(waterVertexBuffer);
            device.DrawUserPrimitives<VertexPositionColorTexture>(PrimitiveType.TriangleStrip, vpnt, 0, 2);

        }
    }
}
```